

# Shading and Diffuse Reflection for Advanced Microprocessor 3D

## Project 3D Graphics Engine Design

### CMPE 240

**Name:** Harish Marepalli

**SJSU ID:** 016707314

**Email:** [harish.marepalli@sjsu.edu](mailto:harish.marepalli@sjsu.edu)

**Team Name:** Student Group\_1

**Team Members:** Tirumala Saiteja Goruganthu, Debasish Panigrahi, Shahnawaz Idariya

**Target Board:** LPC1769

### INTRODUCTION:

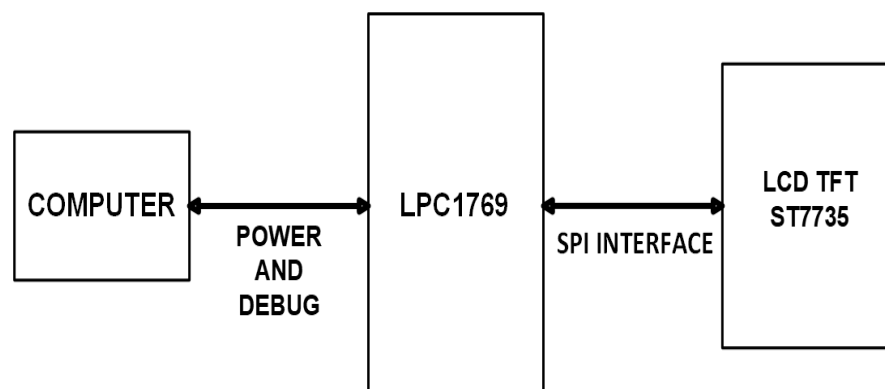
This project's objective is to perform interfacing of the peripheral LCD display module to the NXP LPC1769 board utilizing the Serial Peripheral Interface. The program is designed in such a way that the LPC1769 controls the pixel content displayed on the LCD. The program is implemented in 'C' programming language which displays a rotated 3D cube with a light source and a shadow on the LCD with diffuse reflection shading model. The diffuse reflection is implemented on the top surface of the cube, and it is also implemented for the half sphere. Successful validation of 3D graphics processing engine design is shown on the LPC1769 LCD Module.

This report provides detailed description of the software and hardware methodology implemented in the lab project. Along with this, it also includes details on the testing and verification. Diagrams and Images are also provided when needed to show a clear perspective or output.

The LPC module is connected to the computers USB port to get the required power supply. The LCD module draws in 3.3V from the LPC in built power supply. In LPC1769, pin numbers 11,12,13,14 are used for communication as we are using SPI port number 0 for our project code. The LCD peripheral is connected via appropriate pins with the LPC module's SPI port for communication.

### 1. System block diagram of the entire setup including laptop computer

The figure 1 shown below is the top-level system block diagram or system layout including laptop computer. The power is supplied from the computer to LPC module via a USB cable. The LCD display module is powered from the VCC and GND pins from the LPC1769 module. The LCD display is connected to the LPC GPIO ports via connecting wires using SPI interface.



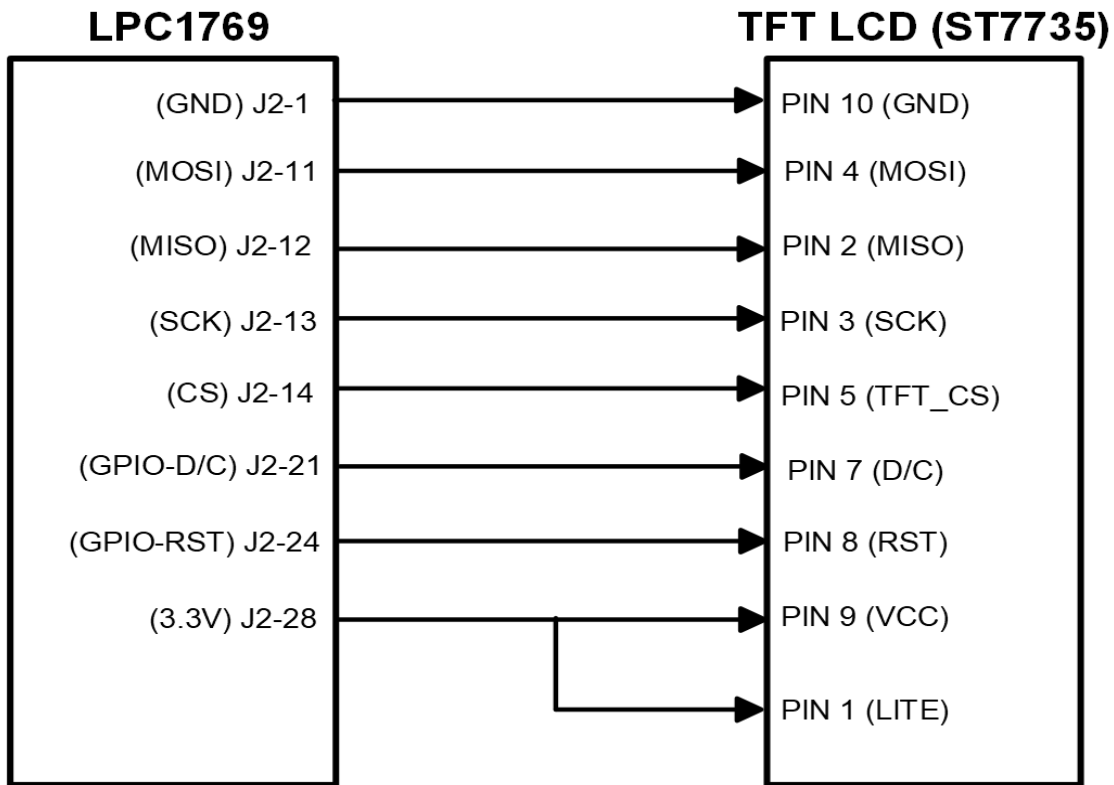
**Fig. 1: System block diagram of the entire setup including laptop computer**

### 1a. Items Used

- i. CPU Module
- ii. LCD Display
- iii. PC

## 2. System block diagram of the SPI color LCD interface

The Figure 2 shown below is the system block diagram of the SPI color LCD interface. The SPI interface circuit is inbuilt on the wire wrapping board of a certain size. It consists of an external LCD and LPC1769 module. The circuit is connected to the computer using the USB cable.



**Fig. 2: System block diagram of the SPI color LCD interface**

## 3. Schematics of the LPC1769 interface to LCD color display panel

The figure 3 below is the schematics of the LPC1769 interfacing with LCD color display panel. LPCXpresso module is connected to the 1.8" 18-bit color TFT LCD display to ensure the functionality working as expected. This module is used to display images and shapes of geometric figures. It uses 4-wire SPI to communicate and has its own pixel addressable frame buffer. The resolution of the LCD display is 128 by 160. It includes a 3.3V regulator for low voltage drop out and a 3/5 level shifter circuit so that we can have input power logic of 3V or 5V. The LPC module and LCD display are built on the wire wrapping board and are connected to the required pin via connecting wires. Once the required hardware is built, MCUXpresso IDE is utilized to program the LPC module.

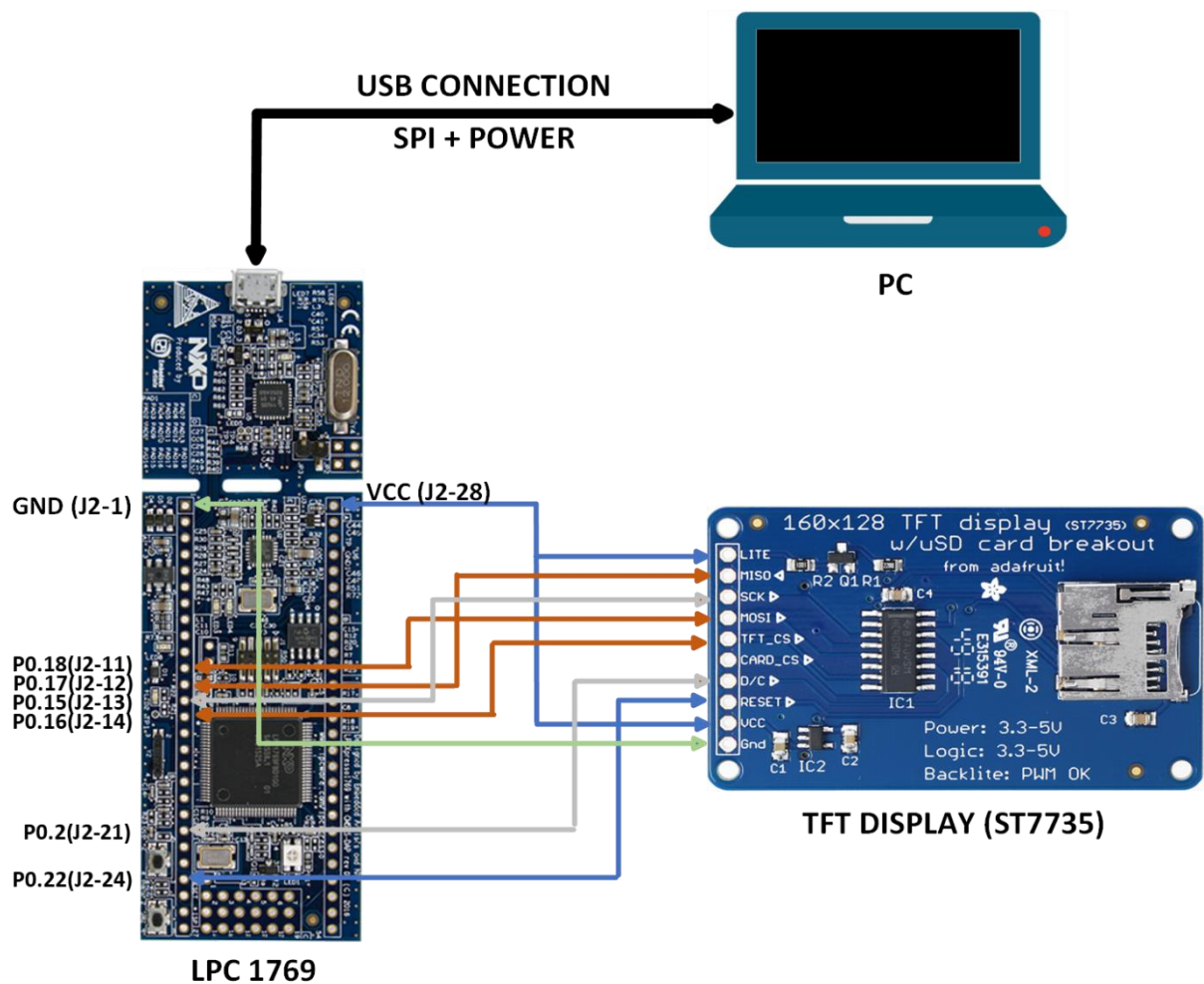


Fig. 3: Schematics of the LPC1769 interface to LCD color display panel

### 3a. LPC1769 PINOUT

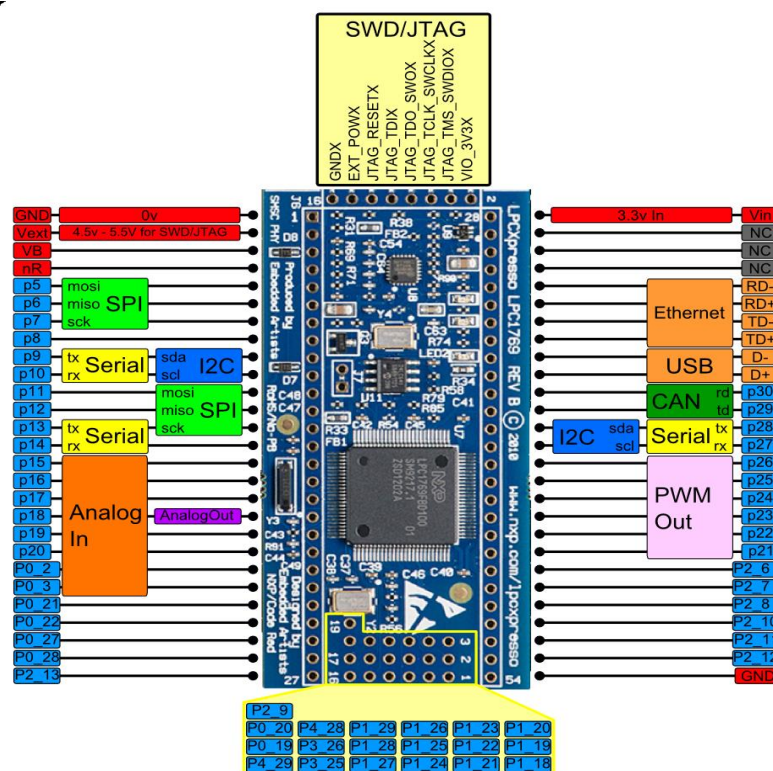
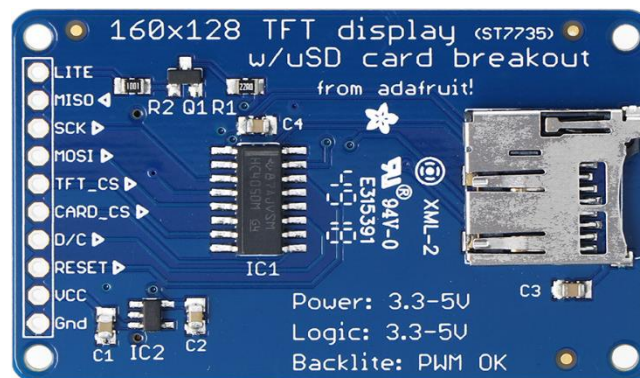


Fig. 3a: LPC1769 PINOUT

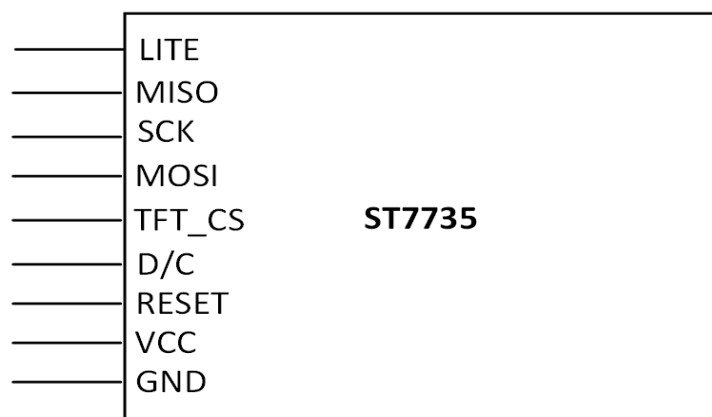
Figure 3a shows the LPC1769 pinout diagram.

### 3b. LCD display and its pinout

Figure 3b shows the LCD display and Figure 3c shows its pinout. The ST7735 is a single chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source line and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI) 8, 12, 16, and 18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption.



**Fig. 3b: 160x128 TFT Display with ST7735**



**Fig. 3c: LCD Pinout**

Some of the technical specifications for the Liquid Crystal Display Module are:

- 1.8" diagonal LCD TFT display.
- 128x160 resolution.
- 18-bit (262,144) color.
- 4 or 5 wire SPI digital interface.
- 2 white LED backlight with PWM dimmer.
- 3.3V/5V compatible power logic.
- Overall dimensions: 34mm x 56mm x 6.5mm.
- Current drawn: 50mA (full backlight)
- Manufactured by Adafruit.
- ST7735 controlled with built in pixel-addressable RAM buffer video.

#### 4. Pin connectivity table

We setup the connections between the LCD and LPC module. The SPI interface circuit is inbuilt on the wire wrapping board of a certain size. It consists of an external LCD and an LPC1769 module. The circuit is connected to the computer using the USB cable. The connections are shown in the table 1 below.

CPU MODULE	TFT LCD DISPLAY
+3V3/J2-28	LITE
MISO0/P0.17/J2-12	MISO
SCK0/P0.15/J2-13	SCK
MOSI0/P0.18/J2-11	MOSI
SSEL0/P0.16/J2-14	TFT_CS
NA	CARD_CS
P0.2/J2-21	D/C
P0.22/J2-24	RESET
+3V3/J2-28	VCC
GND/J2-1	GND

**Table 1: Pin Connectivity between LPC and LCD**

#### Additional Points:

##### Software Used:

1. Coding is done in which data is sent and processed by LCD in the form of frames.
2. Every frame consists of a start of a frame and end of the frame for the LCD to differentiate the incoming data.
3. Usually, the sequence would start with a header byte followed by the command and then the data.
4. At the end, a frame tail would be present.
5. On the whole, the data in each frame is limited to 249 bytes maximum as the total including the header, command, and tail will come up to 255 bytes.
6. The program is built using Adafruit graphics library and MCUXpresso IDE.
7. The software requirements involve initializing the LCD module using Adafruit library for sending data buffer to the LCD as well as polling the GPIO pins for any external interrupt and based on the external interrupt taking actions for displaying content on the LCD screen.
8. We need the following to fulfil the software requirement for successful implementation:
  - a. Windows
  - b. MCUXpresso IDE Version 11
  - c. External LCD Communication Programming Opcode.

## 5. How to compile, run the code, and implement

1. Firstly, go through the theory and formulae taught in the class.
2. Next comes the implementation part. The formulae have to be written in the C language format.
3. First of all, a half sphere is drawn at the center of the coordinate system by using the circle equation. All the circles are drawn one above the other to represent the shape like a half sphere.
4. A few numbers of contours are drawn using the formulae on the half sphere.
5. Then, a cube is drawn by rotating it with respect to the arbitrary vector by an angle of 5 degrees clockwise. Since, it is clockwise, consider the angle to be -5 degrees.
6. This arbitrary vector rotation of the cube involves a total of 7 steps. These 7 steps cover the translation, rotation and postprocessing.
7. Diffuse reflection is performed on the top side of the rotated cube by using red color.
8. After this step, compute the ray equation and its intersection with the  $x_w$ - $y_w$  plane. For the top surface of the cube there are 4 ray equations, each of the ray equations forms intersection point on the  $x_w$ - $y_w$  plane.
9. This set of points are to be stored and then produce a shade of dark blue by plotting a polygon. The reflectivity for red is 0.8 and for blue and green are 0.0. I have done it using the equations for shading.
10. When doing this, the diffuse reflection is not properly generated. So, for this, I used scaling linear equation to scale up the diffuse reflection color from 20 to 255.
11. Finally, I have placed a tree on one of the 2 frontal surfaces of the cube by using the formulae/equations used in the assignment of draw tree.
12. After performing the above operations in theory, create a project by selecting LPC1769 board from the wizard.
13. After creating the project, place all the dependent files at the location of the main C file.
14. After writing the code for performing all the above operations in the MCUXpresso editor, save it and build it for checking the presence of any errors.
15. Connect the prototype board consisting of LPC1769 and LCD display module to the system and click on 'Debug' option present at the left side of the software.
16. After doing this, a breakpoint is hit automatically. At this time, click on the 'Resume/Run' symbol on the top.
17. Then, the program runs completely, and we can see the Coordinates, Sphere, Rotated Cube, Diffuse Reflection, Shadow, and a tree on the cube on the LCD module.
18. We can change the values of the camera location, cube etc as per our requirement.

### Formulae:

A few of the formulas used in the above operations are included below.

World-to-viewer transform:

$$\mathbf{T} = \begin{bmatrix} -\sin \theta & \cos \theta & 0 & 0 \\ -\cos \phi \cos \theta & -\cos \phi \sin \theta & \sin \phi & 0 \\ -\sin \phi \cos \theta & -\sin \phi \sin \theta & -\cos \phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can write equations from this matrix.

1.  $x'_i = -\sin \theta * x_i + \cos \theta * y_i$
2.  $y'_i = -\cos \phi * \cos \theta * x_i - \cos \phi * \sin \theta * y_i + \sin \phi * z_i$

$$3. \quad z_i' = -\sin \phi * \cos \theta * x_i - \sin \phi * \cos \theta * y_i - \cos \phi * z_i + \rho$$

Perspective Projection:

$$x_p = x_e \left( \frac{D}{z_e} \right)$$

$$y_p = y_e \left( \frac{D}{z_e} \right)$$

Diffuse Reflection Equation:

$$I_{diff}(x, y, z) = K * \frac{1}{\|r^2\|} * \frac{n * r}{\|n\| * \|r\|}(r_r, g_g, b_b)$$

### Algorithm:

Few steps are needed to control LCD functionality, below is an algorithm for controlling the LCD:

1. Initialize the SPI and the LCD.
2. Set the bit mask for the MOSI, SCK, SEEL pin of the SPI port.
3. Define function opcode for LCD.
4. Send commands to LCD to display desired image.

(OR)

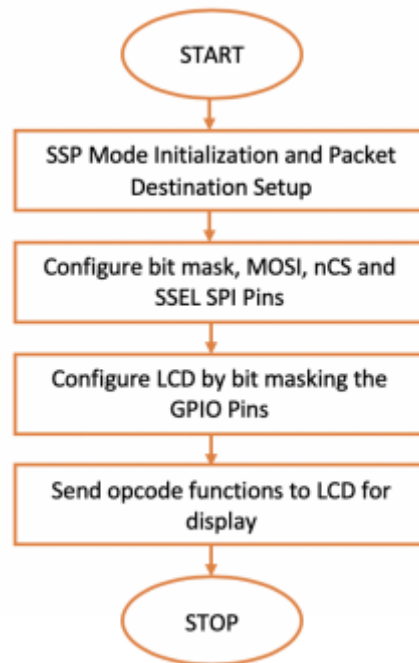
The algorithm specifies the basic steps required to establish the software part in the Integrated Development Environment and making the LCD work as instructed.

1. Make all necessary connections between the Master, Slave and the LCD.
2. Send the data on the MOSI of the LPC, for the device to test.
3. Print the data on the LCD.
4. Select and send the next data/command, and disable the chip select.
5. Send the data which is supposed to be written.
6. Set the chip select.
7. Clear the chip select and display 3D axis on the LCD.
8. Call the function of cube to display on the screen.
9. Draw the cube in XYZ plane of LCD.
10. Draw the shadow of the cube.
11. Decorate the surfaces of cube with colors and half sphere with color.
12. Diffuse reflection on the top surface of the cube with decoration is computed.

### Flowchart:

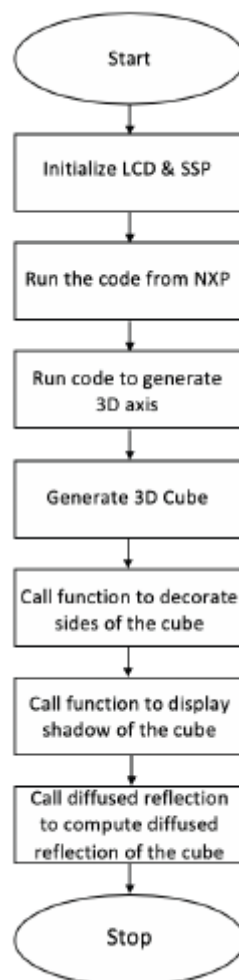
The above steps are represented in the form of a flow chart.

Figure 4 shows the flowchart representation.



**Fig. 4: Program Flowchart**

(OR)



**Fig. 4: Program Flowchart**



## Pseudo Code:

The pseudo code for the LCD is implemented and tested. This code will draw the coordinates, half sphere, rotated cube, diffuse reflection on the top surface of the cube. A shadow is drawn for the rotated cube and diffuse reflection and scaling is implemented for the half sphere. A tree is drawn on one of the frontal faces of the rotated cube. The first step in the procedure is to build the code. Then we should debug the code. Immediately, “Run” action is executed. This makes the code to run in a loop.

Pseudo Code:

```
#include "LPC17xx.h"    //LPC17xx definitions

void spiwrite(uint8_t c) //get the input 'c'

void writecommand(uint8_t c) //get the input 'c'

void writedata(uint8_t c) //get the input 'c' and write data according to it

void writeword(uint16_t c) //get the input 'c' and write data according to it

void write888(uint32_t color, uint32_t repeat) //get the color and repeat parameter to display

void setAddrWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1) //get four parameters

void fillrect(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color) //get four coordinates and
color to display the area for the display

void lcddelay(int ms) //delay the display time in ms

void lcd_init() //initializes the LCD

int16_t xCartesian(int16_t x) //Coordinate to Cartesian (Converting virtual X-Coordinate to physical
X-Coordinate)

int16_t yCartesian(int16_t y) // Converting virtual Y-Coordinate to physical Y-Coordinate

void drawPixel(int16_t x, int16_t y, uint32_t color) //Inputs are x, y, and color. The inputs x, y, and
color get two coordinates

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color) //get four coordinates and
display line according to it

float Lambda3D(float Zi, float Zs) //used to calculate the lambda value by using the formula

Pts3D ShadowPoint3D(Pts3D Pi, Pts3D Ps, float lambda) //used to calculate the 3D points of the
shadow of the rotated cube

int getDiffuseColor(Pts3D Pi) //used to calculate the diffuse reflection for the top surface of the cube

int getDiffuseColorGreen(Pts3D Pi) //used to calculate the diffuse reflection for the half sphere

Pts2D get3DTransform(Pts3D Pi) //used to calculate the 3D transform by using the related
formulas

Pts3D rotate_pointIn3D(Pts3D p, Pts3D o, float angle) //Rotate point p with respect to o and angle

void drawTree(float xstart, float ystart, float zstart, int cube_side) //It is used to draw the tree on of
the frontal surfaces of the rotated cube

void designTreeTrunkIn3D(Pts3D start3D, Pts3D end3D, uint32_t color, uint8_t thickness) //Design
the trunk of a tree
```

```

void designTreeIn3D(Pts3D start3D, Pts3D end3D, int level, double lambda) //Design a Tree using
drawline method and rotatepoint method

Pts3D rotateCoord3D(Pts3D ARBPi, Pts3D ARBPi1, int angle, float cube_x, float cube_y, float cube_z)
//Rotate cube with respect to the arbitrary vector. It involves 7 steps

void drawCube() //Draw the cube, Define given Arbitrary vectors, and Shadow Calculation, world to
viewer transform, Shadow fill, color the top side of the cube with diffuse reflection, color the left sides
of the cube with certain colors are required. Calls the drawTree method

void drawSphere() //Method to draw the half sphere using contours, diffuse reflection for it

int main (void) //main method

```

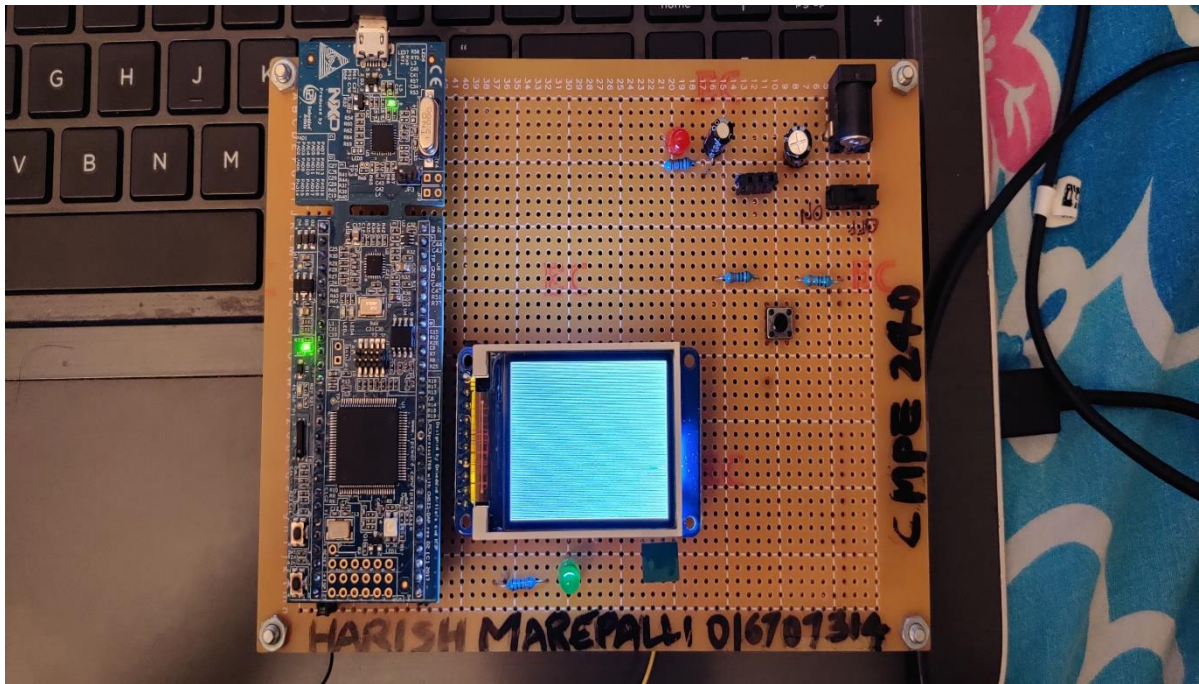
## Testing and Verification

1. Testing section mainly has to deal with testing the code and verifying the outputs.
2. Procedures are followed to ensure working of the LCD.
3. A multi-meter is used to ensure that no components are being shorted. This is done before debugging and actual testing is done.
4. Verify that the CPU is connected and LPC link is established.
5. Verify the LCD is properly connected to LPC.
6. Check to see that LCD is given VCC and GND from LPC1769.
7. IDE needs to recognize the USB link for the program to be debugged and to generate output.
8. When the program is debugged, the LCD module should be initialized and output should be displayed.
9. The LCD glows when the CPU module receives the power.
10. The project if having no errors would build and debug successfully with successful link connected to the CPU module.
11. The data would be transferred from main program in IDE to the CPU via USB cable.
12. This will initiate the display of the cube with 3D axis.
13. 3D Cube shading Model: First, cube was displayed in world coordinate system.
14. Then, a point was created and was treated as a source of light and shadow has been implemented by computing shadow points from source of light point.
15. Next, we have filled the sides displayed in perspective view with color.
16. After that, compute diffuse reflection on the top surface of the cube using formula.
17. The four ray equations derived mathematically. So, keep track each set of 4 points and produce a dark red or such shade for the cube by plotting 4 vertices polygons. Assuming reflective nature for red is 0.8 and for blue and green are 0.0.
18. All this is done in a 3D space, which is then projected onto a 2D space. This 2D space was displayed on the LCD display.
19. The cube is rotated by the given angle value.
20. Shadow is computed for the rotated cube.
21. Diffuse reflection for the half sphere is also computed.
22. A tree is drawn on one of the frontal surfaces of the cube.
23. Scaling is computed for scaling the diffuse reflection.
24. After filling the entire screen, a delay is introduced so that a better view of the 3D display can be obtained.
25. Thus, interfacing of LCD with LPC1769 using SPI can be tested and verified successfully.

## Screen Captures

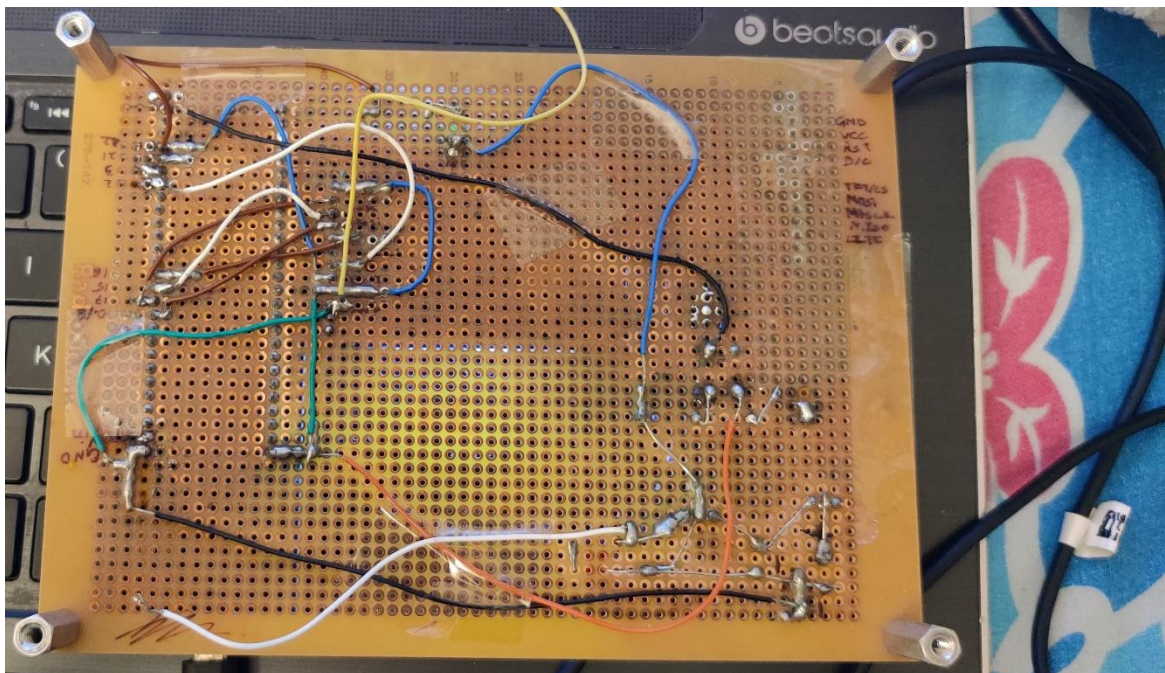
Below are the images of implementation.

1. Figure 5 shows the Prototype board front side. The prototype board is used to connect LPC1769 with the LCD display.



**Fig. 5: Prototype Board Front**

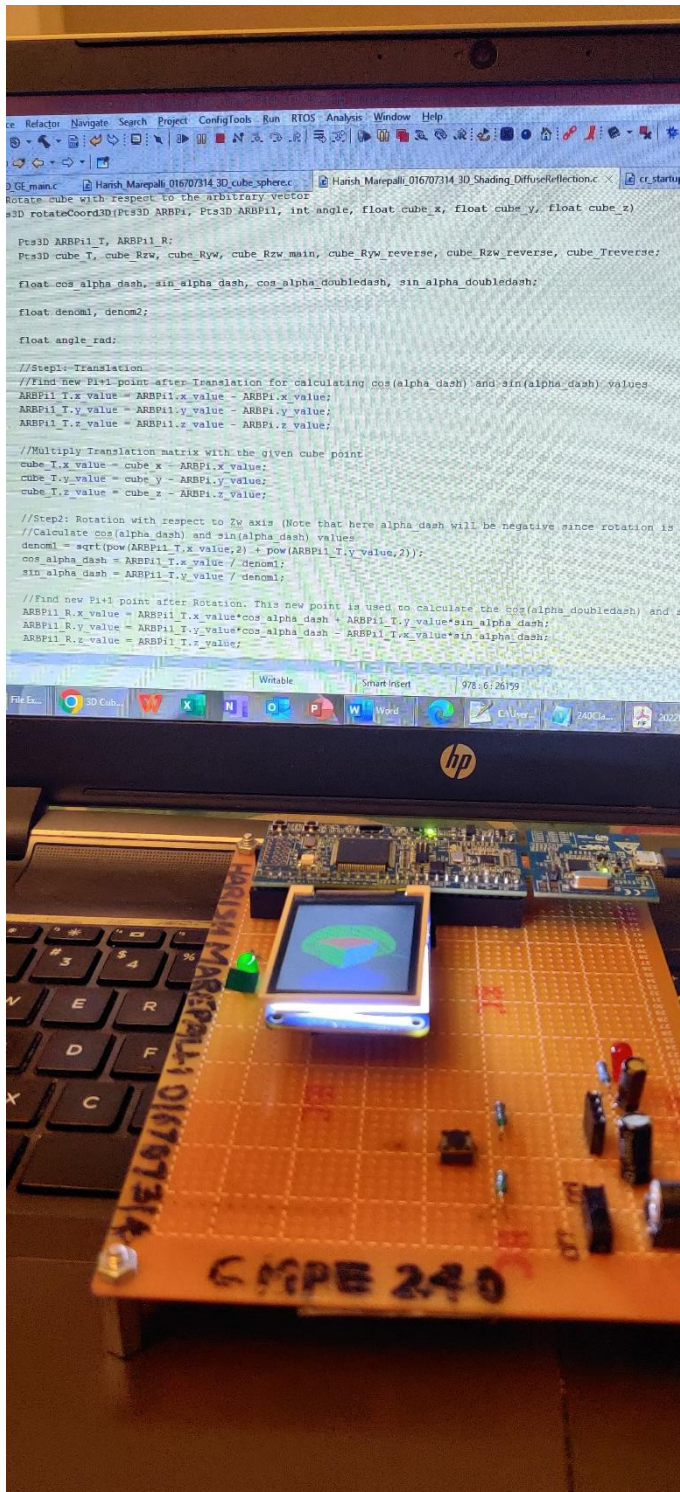
2. Figure 6 shows the Prototype board back side connections.



**Fig. 6: Prototype Board Back**



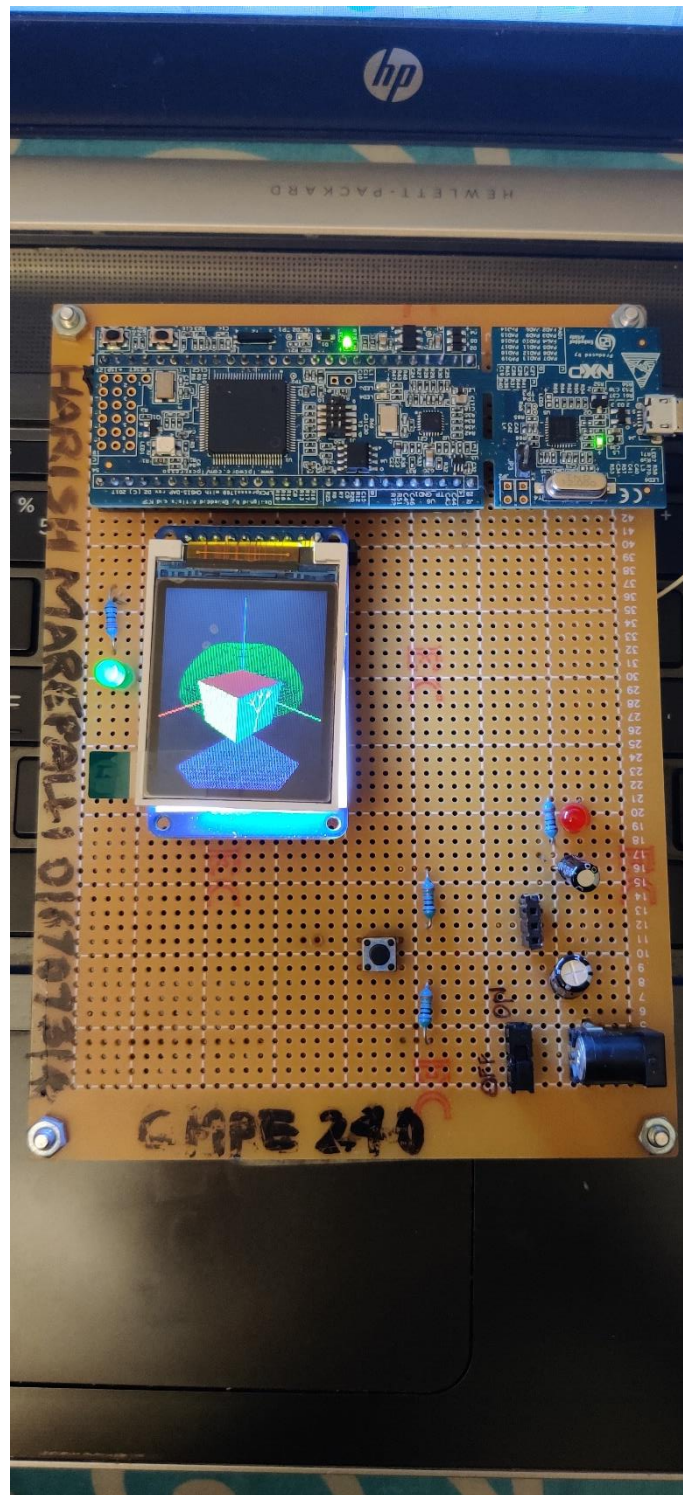
3. Figure 7 shows the laptop with code and the prototype board containing LPC1769 and LCD display.



**Fig. 7: laptop with code and Prototype Board**

- Below figures show the Half Sphere and its diffuse reflection, Rotated Cube and its diffuse reflection of the top surface, Shadow and its diffuse reflection, a tree on one of the frontal faces.

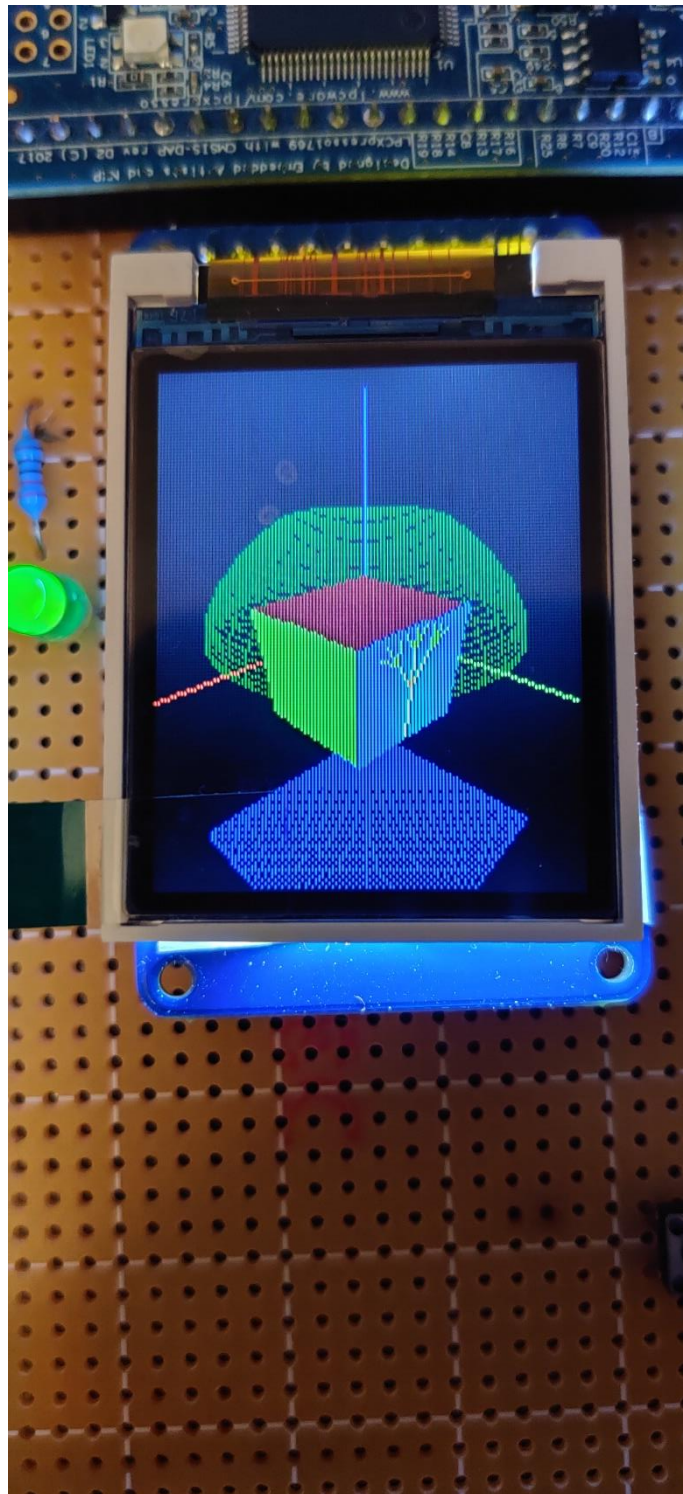
Figure 8 shows the output for Eye coordinates (150, 150, 100) with prototype board and LCD.



**Fig. 8: Output for Eye (150, 150 ,100) with prototype board and LCD**

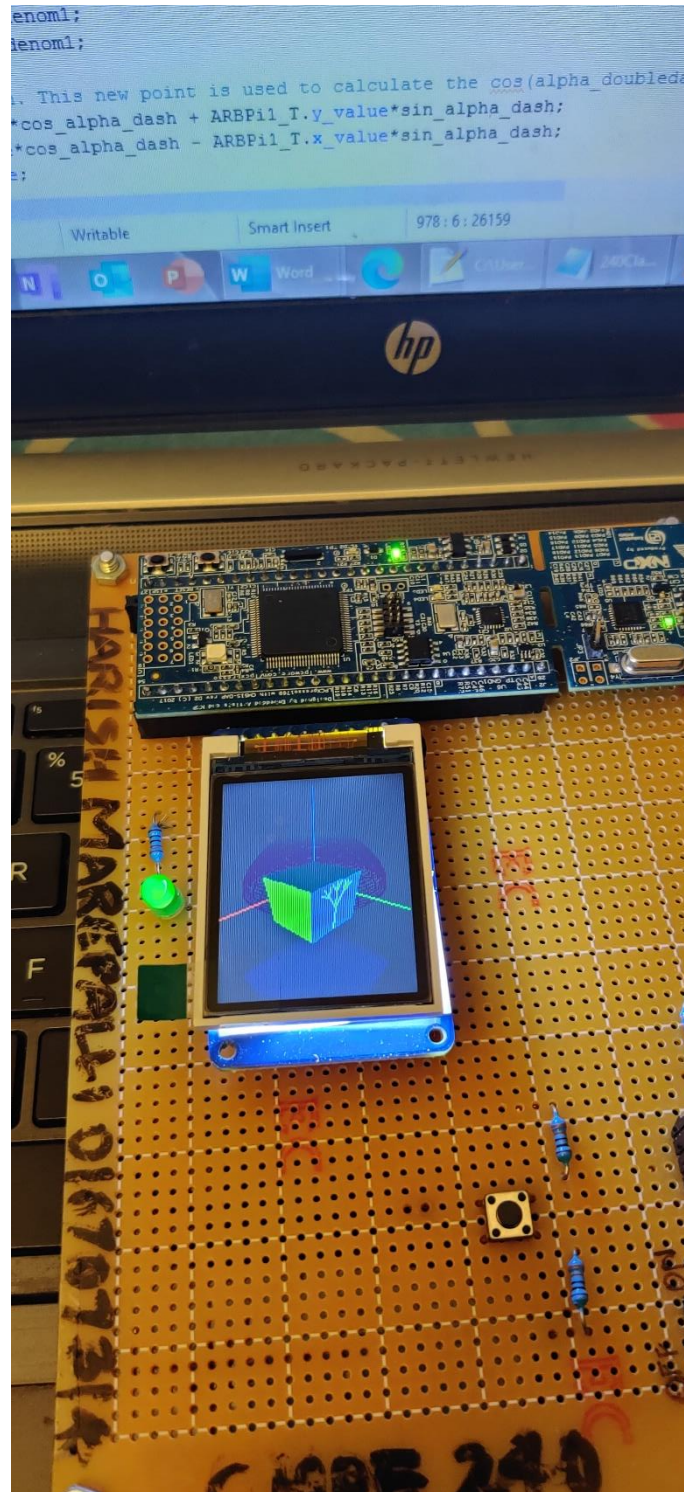


Figure 9 shows the output for Eye coordinates (150, 150, 100) with LCD in the photo.



**Fig. 9: Output for Eye (150, 150, 100) with LCD**

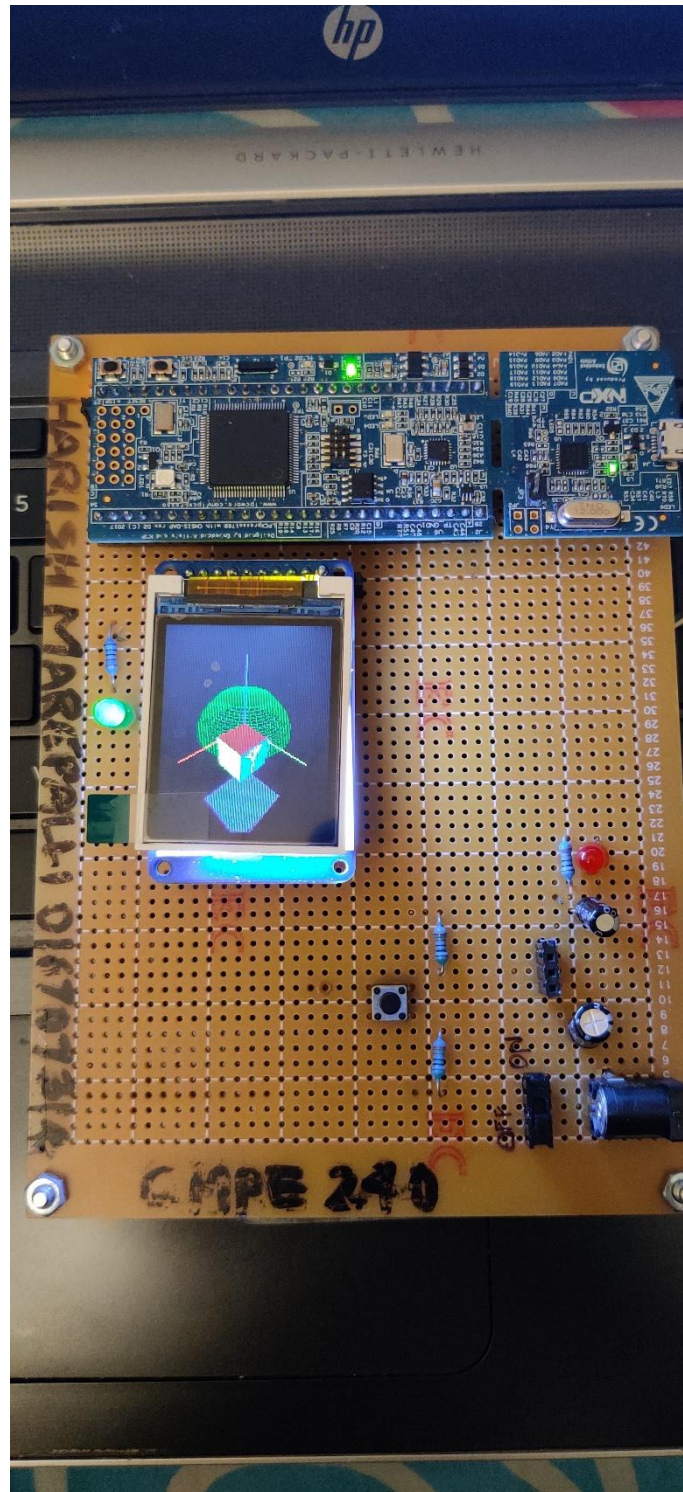
Figure 10 shows the output for Eye (150, 150, 100) by changing the angle of the phone to view diffusion reflection.



**Fig. 10: Output for Eye (150, 150, 100) by changing the angle of the phone**



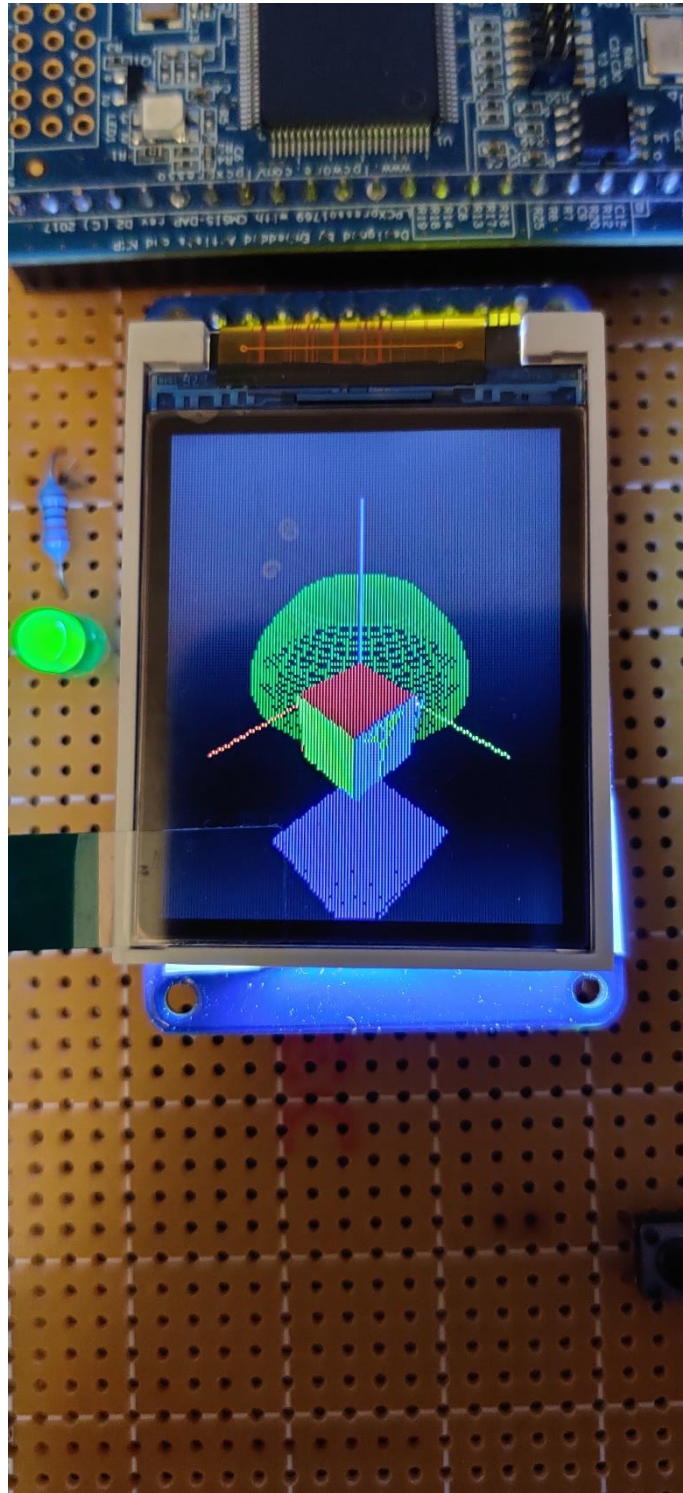
Figure 11 shows the output for Eye coordinates (200, 200, 200) with prototype board and LCD.



**Fig. 11: Output for Eye (200, 200, 200) with prototype board and LCD**

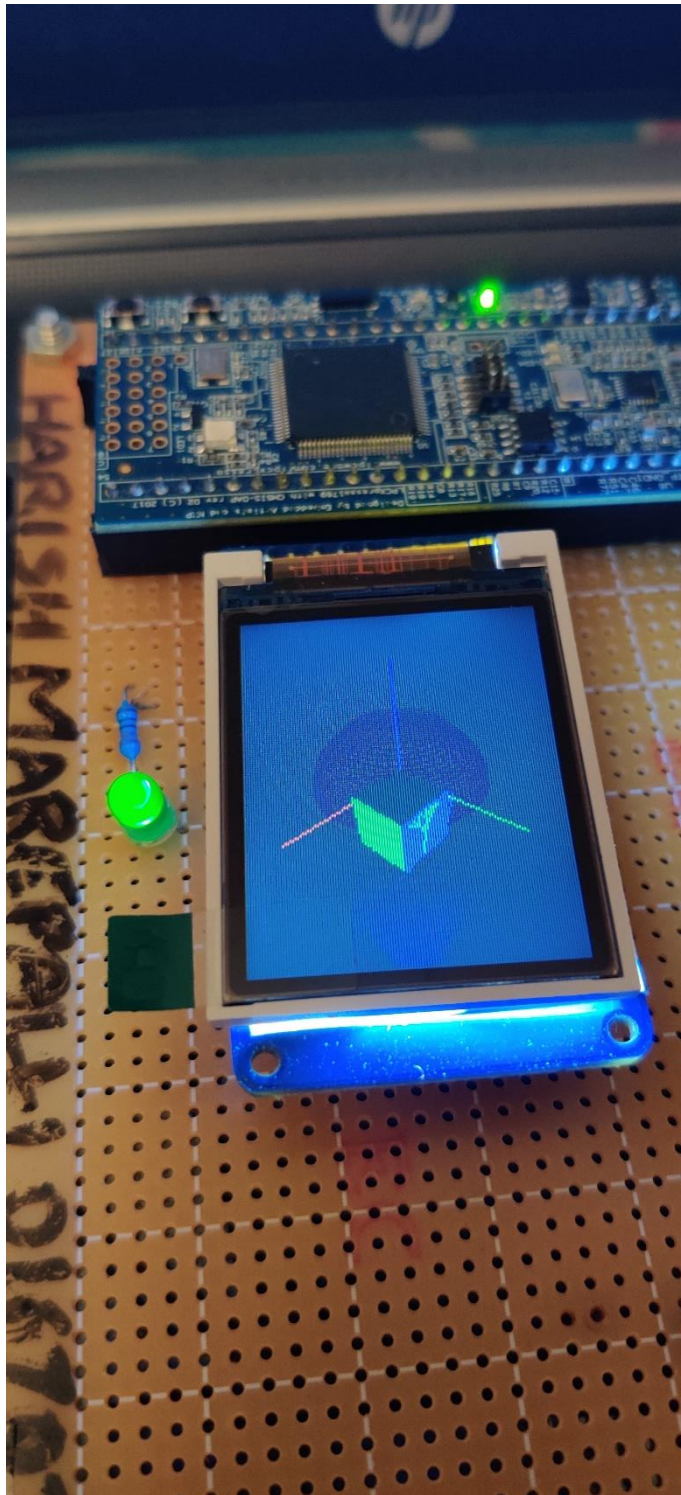


Figure 12 shows the output for Eye coordinates (150, 150, 100) with LCD in the photo.



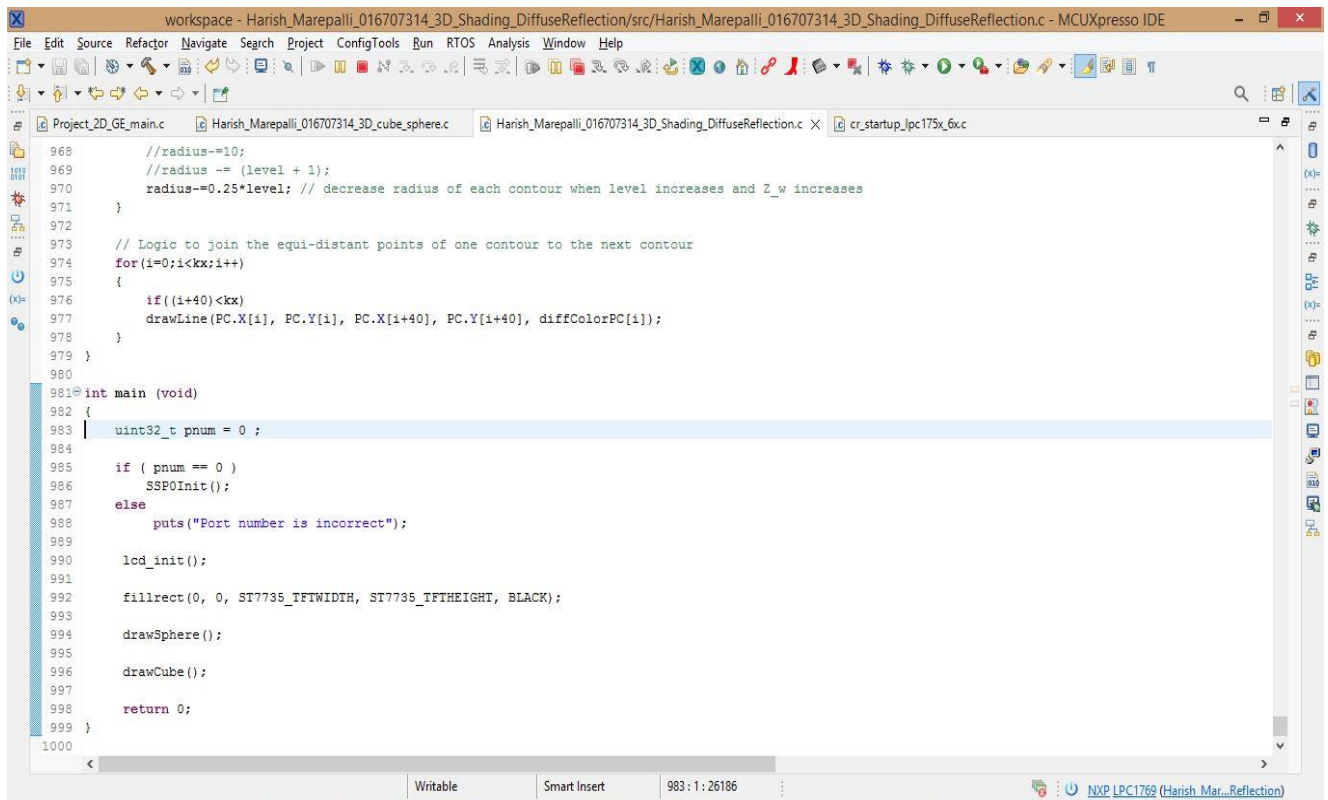
**Fig. 12: Output for Eye (200, 200, 200) with LCD**

Figure 13 shows the output for Eye (200, 200, 200) by changing the angle of the phone to view diffusion reflection.



**Fig. 13:** Output for Eye (200, 200, 200) by changing the angle of the phone

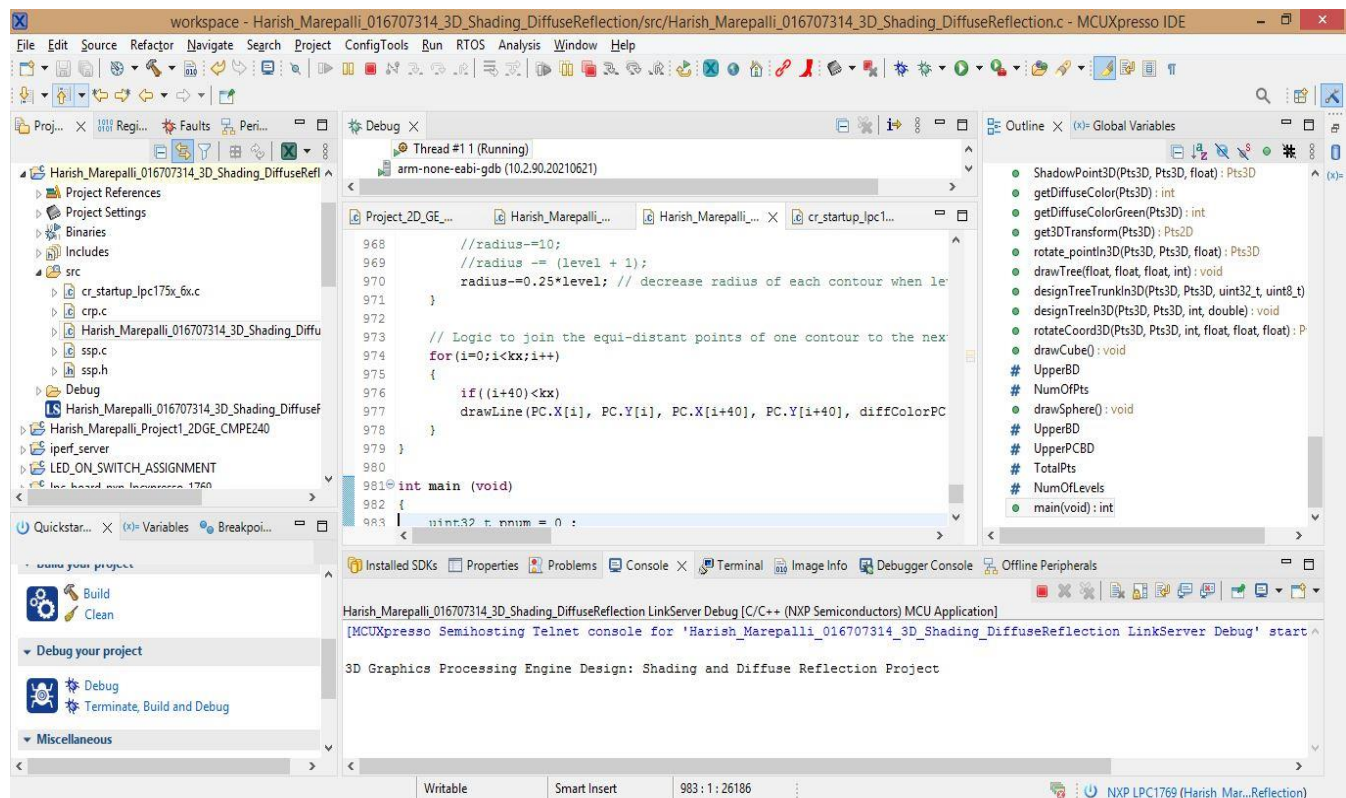
Figure 14 shows the code snippet with only the editor.



```
workspace - Harish_Marepalli_016707314_3D_Shading_DiffuseReflection/src/Harish_Marepalli_016707314_3D_Shading_DiffuseReflection.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Project_2D_GE_main.c Harish_Marepalli_016707314_3D_cube_sphere.c Harish_Marepalli_016707314_3D_Shading_DiffuseReflection.c X cr_startup_lpc175x_6xx
968 //radius-=10;
969 //radius -= (level + 1);
970 radius-=0.25*level; // decrease radius of each contour when level increases and Z_w increases
971 }
972
973 // Logic to join the equi-distant points of one contour to the next contour
974 for(i=0;i<kx;i++)
975 {
976     if((i+40)<kx)
977         drawLine(PC.X[i], PC.Y[i], PC.X[i+40], PC.Y[i+40], diffColorPC[i]);
978 }
979 }
980
981 int main (void)
982 {
983     uint32_t pnum = 0 ;
984
985     if ( pnum == 0 )
986         SSP0Init();
987     else
988         puts("Port number is incorrect");
989
990     lcd_init();
991
992     fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);
993
994     drawSphere();
995
996     drawCube();
997
998     return 0;
999 }
1000
Writable Smart Insert 983 : 1:26186 NXP LPC1769 (Harish_Mar...Reflection)
```

Fig. 14: Code Editor

Figure 15 shows the code snippet along with console.



```
workspace - Harish_Marepalli_016707314_3D_Shading_DiffuseReflection/src/Harish_Marepalli_016707314_3D_Shading_DiffuseReflection.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Proj... X Regi... X Faults X Peri... X
Harish_Marepalli_016707314_3D_Shading_DiffuseRef...
  Project References
  Project Settings
  Binaries
  Includes
  src
    cr_startup_lpc175x_6xx.c
    crp.c
    Harish_Marepalli_016707314_3D_Shading_Diffu...
    ssp.c
    ssp.h
  Debug
    Harish_Marepalli_016707314_3D_Shading_DiffuseF...
  Harish_Marepalli_Project1_2DGE_CMPE240
  LED_ON_SWITCH_ASSIGNMENT
  Use board run frequency: 1760
Quickstar... X Variables Breakpoi...
Build Clean
Debug your project
  Debug
  Terminate, Build and Debug
Miscellaneous
Thread #1:1 (Running)
arm-none-eabi-gdb (10.2.90.20210621)
Project_2D_GE... Harish_Marepalli... Harish_Marepalli... X cr_startup_lpc1...
968 //radius-=10;
969 //radius -= (level + 1);
970 radius-=0.25*level; // decrease radius of each contour when le
971 }
972
973 // Logic to join the equi-distant points of one contour to the nex
974 for(i=0;i<kx;i++)
975 {
976     if((i+40)<kx)
977         drawLine(PC.X[i], PC.Y[i], PC.X[i+40], PC.Y[i+40], diffColorPC
978 }
979 }
980
981 int main (void)
982 {
983     uint32_t pnum = 0 ;
984
985     if ( pnum == 0 )
986         SSP0Init();
987     else
988         puts("Port number is incorrect");
989
990     lcd_init();
991
992     fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);
993
994     drawSphere();
995
996     drawCube();
997
998     return 0;
999 }
1000
Outline X Global Variables
ShadowPoint3D(Pts3D, Pts3D, float) : Pts3D
getDiffuseColor(Pts3D) : int
getDiffuseColorGreen(Pts3D) : int
get3DTransform(Pts3D) : Pts3D
rotate_pointln3D(Pts3D, Pts3D, float) : Pts3D
drawTree(float, float, float, int) : void
designTreeTrunkln3D(Pts3D, Pts3D, uint32_t, uint8_t)
designTreeIn3D(Pts3D, Pts3D, int, double) : void
rotateCoord3D(Pts3D, Pts3D, int, float, float, float) : P
drawCube() : void
# UpperBD
# NumOfPts
drawSphere() : void
# UpperBD
# UpperPCBD
# TotalPts
# NumOfLevels
main(void) : int
Harish_Marepalli_016707314_3D_Shading_DiffuseReflection LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
[MCUXpresso Semihosting Telnet console for 'Harish_Marepalli_016707314_3D_Shading_DiffuseReflection LinkServer Debug' start ^
3D Graphics Processing Engine Design: Shading and Diffuse Reflection Project
Writable Smart Insert 983 : 1:26186 NXP LPC1769 (Harish_Mar...Reflection)
```

Fig. 16: Code with Console

## **CONCLUSION**

The graphics engine design was tested successfully by displaying tree and 3D cube on the hardware connecting the LCD module with the LPC1769. The work provides an opportunity to study 7805 voltage regulators, SPI LCD displays, LPC CPU module. I have done the Shading model and debugged on MCUXpresso IDE.



## APPENDIX

### SOURC CODE

```
/*
=====
Name       : Harish_Marepalli_016707314_3D_Shading_DiffuseReflection.c
Author      : Harish Marepalli
Version     :
Copyright   : $(copyright)
Description : 1. This program is used for creating and displaying the world coordinate
system, a 3D cube and a half sphere object based on the transformation pipeline.
              2. It is also used to rotate the cube w.r.t an arbitrary vector.
              3. Used to compute diffuse reflection on the top surface of the cube.
              4. Used to place a point light source in the world coordinate system
and produce a shadow of dark blue by plotting a polygon.
              5. Used to scale up the diffuse reflection color by using linear
equation.
              6. Used to place a tree on one of the 2 frontal surfaces of the cube.
              7. Used to compute diffuse reflection on visible part of the half
sphere.
=====
*/

#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"           /* LPC17xx definitions */
#include "ssp.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Be careful with the port number and location number, because some of the locations may
not exist in that port. */

#define PORT_NUM          0

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29

#define swap(x, y) {x = x + y; y = x - y; x = x - y ;}

// Defining Color Values
#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF

// Custom Defined Colors
```

```

#define BROWN 0xA52A2A
#define YELLOW 0xFFFE00

// Custom Defined GREEN Shades
#define GREEN1 0x38761D
#define GREEN2 0x002200
#define GREEN3 0x00FC7C
#define GREEN4 0x32CD32
#define GREEN5 0x228B22
#define GREEN6 0x006400

// Custom Defined RED Shades
#define RED1 0xE61C1C
#define RED2 0xEE3B3B
#define RED3 0xEF4D4D
#define RED4 0xE88080

#define GREY1 0x7A7C7B

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

#define pi 3.1416

typedef struct Point
{
    float x;
    float y;
}Point;

// Structure for 2D
typedef struct
{
    float x; float y;
}Pts2D;

// Structure for 3D
typedef struct
{
    float x_value; float y_value; float z_value;
}Pts3D;

void spiwrite(uint8_t c)
{
    int pnum = 0;

    src_addr[0] = c;

    SSP_SSELToggle( pnum, 0 );

    SSPSend( pnum, (uint8_t *)src_addr, 1 );

    SSP_SSELToggle( pnum, 1 );
}

void writecommand(uint8_t c)
{
    LPC_GPIO0->FIOCLR |= (0x1<<2);

    spiwrite(c);
}

void writedata(uint8_t c)
{
    LPC_GPIO0->FIOSET |= (0x1<<2);

    spiwrite(c);
}

```

```

}

void writeword(uint16_t c)
{
    uint8_t d;

    d = c >> 8;

    writedata(d);

    d = c & 0xFF;

    writedata(d);
}

void write888(uint32_t color, uint32_t repeat)
{
    uint8_t red, green, blue;

    int i;

    red = (color >> 16);

    green = (color >> 8) & 0xFF;

    blue = color & 0xFF;

    for (i = 0; i < repeat; i++)
    {
        writedata(red);

        writedata(green);

        writedata(blue);
    }
}

void setAddrWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
{
    writecommand(ST7735_CASET);

    writeword(x0);

    writeword(x1);

    writecommand(ST7735_RASET);

    writeword(y0);

    writeword(y1);
}

void fillrect(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
    //int16_t i;

    int16_t width, height;

    width = x1-x0+1;

    height = y1-y0+1;

    setAddrWindow(x0,y0,x1,y1);

    writecommand(ST7735_RAMWR);

    write888(color,width*height);
}

```

```

}

void lcddelay(int ms)
{
    int count = 24000;

    int i;

    for ( i = count*ms; i > 0; i--);
}

void lcd_init()
{
    int i;
    printf("3D Graphics Processing Engine Design: Shading and Diffuse Reflection
Project\n");
    // Set pins P0.16, P0.2, P0.22 as output
    LPC_GPIO0->FIODIR |= (0x1<<16);

    LPC_GPIO0->FIODIR |= (0x1<<2);

    LPC_GPIO0->FIODIR |= (0x1<<22);

    // Hardware Reset Sequence
    LPC_GPIO0->FIOSET |= (0x1<<22);
    lcddelay(500);

    LPC_GPIO0->FIOCLR |= (0x1<<22);
    lcddelay(500);

    LPC_GPIO0->FIOSET |= (0x1<<22);
    lcddelay(500);

    // initialize buffers
    for ( i = 0; i < SSP_BUFSIZE; i++ )
    {
        src_addr[i] = 0;
        dest_addr[i] = 0;
    }

    // Take LCD display out of sleep mode
    writecommand(ST7735_SLPOUT);
    lcddelay(200);

    // Turn LCD display on
    writecommand(ST7735_DISPON);
    lcddelay(200);
}

// Coordinate to Cartesian (Converting virtual X-Coordinate to physical X-Coordinate)
int16_t xCartesian(int16_t x)
{
    x = x + (_width>>1);
    return x;
}

// Converting virtual Y-Coordinate to physical Y-Coordinate
int16_t yCartesian(int16_t y)
{
    y = (_height>>1) - y;
    return y;
}

void drawPixel(int16_t x, int16_t y, uint32_t color)
{
    // Cartesian Implementation
    x=xCartesian(x); y=yCartesian(y);

```



```

    if ((x < 0) || (x >= _width) || (y < 0) || (y >= _height))

    return;

    setAddrWindow(x, y, x + 1, y + 1);

    writecommand(ST7735_RAMWR);

    write888(color, 1);
}

/*****

** Descriptions:          Draw line function
**
** parameters:            Starting point (x0,y0), Ending point(x1,y1) and color
** Returned value:        None
**

*****/

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
    int16_t slope = abs(y1 - y0) > abs(x1 - x0);

    if (slope)
    {
        swap(x0, y0);
        swap(x1, y1);
    }

    if (x0 > x1)
    {
        swap(x0, x1);
        swap(y0, y1);
    }

    int16_t dx, dy;

    dx = x1 - x0;

    dy = abs(y1 - y0);

    int16_t err = dx / 2;

    int16_t ystep;

    if (y0 < y1)
    {
        ystep = 1;
    }

    else
    {
        ystep = -1;
    }

    for (; x0 <= x1; x0++)
    {

```

```

        if (slope)
        {
            drawPixel(y0, x0, color);
        }

        else
        {
            drawPixel(x0, y0, color);
        }

        err -= dy;

        if (err < 0)
        {
            y0 += ystep;

            err += dx;
        }
    }
}

float Lambda3D(float Zi, float Zs)
{
    float lambda;
    lambda = -Zi/(Zs-Zi);
    return lambda;
}

Pts3D ShadowPoint3D(Pts3D Pi, Pts3D Ps, float lambda)
{
    Pts3D pt;
    pt.x_value = (Pi.x_value + (lambda*(Ps.x_value-Pi.x_value)));
    pt.y_value = (Pi.y_value + (lambda*(Ps.y_value-Pi.y_value)));
    pt.z_value = (Pi.z_value + (lambda*(Ps.z_value-Pi.z_value)));
    return pt;
}

int getDiffuseColor(Pts3D Pi)
{
    uint32_t print_diffuse_color;
    Pts3D Ps = {-20,-20,220};
    int diff_red, diff_green, diff_blue;
    float new_red, new_green, new_blue, r1=0.8, r2=0.0, r3=0.0, scaling = 16000;

    float_t temp = ((Ps.z_value - Pi.z_value)/sqrt(pow((Ps.x_value - Pi.x_value),2) +
pow((Ps.y_value - Pi.y_value),2) + pow((Ps.z_value - Pi.z_value),2))) / (pow((Ps.x_value -
Pi.x_value),2) + pow((Ps.y_value - Pi.y_value),2) + pow((Ps.z_value - Pi.z_value),2));

    temp *= scaling;
    diff_red = r1 * temp * 255;
    diff_green = r2 * temp;
    diff_blue = r3 * temp;
    new_red = (diff_red << 16);
    new_green = (diff_green << 8);
    new_blue = (diff_blue);
    print_diffuse_color = new_red + new_green + new_blue;
    return print_diffuse_color;
}

int getDiffuseColorGreen(Pts3D Pi)
{
    uint32_t print_diffuse_color;
    Pts3D Ps = {-20,-20,220};
    int diff_red, diff_green, diff_blue;
    float new_red, new_green, new_blue, r1=0.0, r2=1.0, r3=0.0, scaling = 8000;

```

```

    float_t temp = ((Ps.z_value - Pi.z_value)/sqrt(pow((Ps.x_value - Pi.x_value),2) +
pow((Ps.y_value - Pi.y_value),2) + pow((Ps.z_value - Pi.z_value),2))) / (pow((Ps.x_value -
Pi.x_value),2) + pow((Ps.y_value - Pi.y_value),2) + pow((Ps.z_value - Pi.z_value),2));

    temp *= scaling;
    diff_red = r1 * temp * 255;
    diff_green = r2 * temp * 255;
    diff_blue = r3 * temp * 255;
    new_red = (diff_red << 16);
    new_green = (diff_green << 8);
    new_blue = (diff_blue);
    print_diffuse_color = new_red + new_green +new_blue;
    return print_diffuse_color;
}

Pts2D get3DTransform(Pts3D Pi)
{
    float Xe=150, Ye=150, Ze=100;
    float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));
    float_t D_focal=120;

    typedef struct{
        float X; float Y; float Z;
    }pworld;

    typedef struct{
        float X; float Y; float Z;
    }pviewer;

    typedef struct{
        float X; float Y;
    }pperspective;

    pworld world;
    pviewer viewer;
    pperspective perspective;

    world.X = Pi.x_value;
    world.Y = Pi.y_value;
    world.Z = Pi.z_value;

    float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
    float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
    float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
    float cPhi = Ze/Rho;

    viewer.X=-sPheta*world.X+cPheta*world.Y;
    viewer.Y = -cPheta * cPhi * world.X - cPhi * sPheta * world.Y + sPhi * world.Z;
    viewer.Z = -sPhi * cPheta * world.X - sPhi * cPheta * world.Y-cPheta * world.Z + Rho;

    perspective.X=D_focal*viewer.X/viewer.Z;
    perspective.Y=D_focal*viewer.Y/viewer.Z;

    Pts2D pt;
    pt.x=perspective.X;
    pt.y=perspective.Y;
    return pt;
}

/* Rotate point p with respect to o and angle <angle> */
Pts3D rotate_pointIn3D(Pts3D p, Pts3D o, float angle)
{
    Pts3D rt, t, new;

    float s = sin(angle);

```

```

    float c = cos(angle);

    //translate point to origin
    /*t.x_value = 105.00;
    t.y_value = p.y_value - o.y_value;
    t.z_value = p.z_value - o.z_value;

    rt.x_value = 105.00;
    rt.y_value = t.y_value * c - t.z_value * s;
    rt.z_value = t.y_value * s + t.z_value * c;

    //translate point back
    new.x_value = 105.00;
    new.y_value = rt.y_value + o.y_value;
    new.z_value = rt.z_value + o.z_value;*/

    //translate point to origin
    t.x_value = p.x_value - o.x_value;
    t.y_value = o.y_value;
    t.z_value = p.z_value - o.z_value;

    rt.x_value = t.x_value * c - t.z_value * s;
    rt.y_value = o.y_value;
    rt.z_value = t.x_value * s + t.z_value * c;

    //translate point back
    new.x_value = rt.x_value + o.x_value;
    new.y_value = o.y_value;
    new.z_value = rt.z_value + o.z_value;

    return new;
}

void drawTree(float xstart, float ystart, float zstart, int cube_side)
{
    Pts2D lcd, start, end, c;
    Pts3D start3D, end3D, c3D;

    double lambda = 0.6;

    /*start3D.x_value = xstart + cube_side;
    start3D.y_value = ystart + (cube_side/2);
    start3D.z_value = zstart;

    end3D.x_value = xstart + cube_side;
    end3D.y_value = ystart + (cube_side/2);
    end3D.z_value = zstart + (cube_side/2);*/

    start3D.x_value = xstart + (cube_side/2);
    start3D.y_value = ystart + cube_side;
    start3D.z_value = zstart;

    end3D.x_value = xstart + (cube_side/2);
    end3D.y_value = ystart + cube_side;
    end3D.z_value = zstart + (cube_side/2);

    designTreeTrunkIn3D(start3D, end3D, GREEN, 1);
    designTreeIn3D(start3D, end3D, 2, lambda);
}

/* Design the trunk of a tree */
void designTreeTrunkIn3D(Pts3D start3D, Pts3D end3D, uint32_t color, uint8_t thickness)
{
    Pts2D lcd, start, end;

    lcd = get3DTransform(start3D);

```

```

start.x = lcd.x;
start.y = lcd.y;

lcd = get3DTransform(end3D);

end.x = lcd.x;
end.y = lcd.y;

int i;
for(i = 0; i < thickness; i++)
    drawLine(start.x + i, start.y, end.x + i, end.y, color);
}

/* Design a Tree using drawline method and rotatepoint method */
void designTreeIn3D(Pts3D start3D, Pts3D end3D, int level, double lambda)
{
    Pts2D lcd, c, start, end;
    Pts3D c3D, rtl3D, rtr3D;

    if(level == 0)
        return;

    lcd = get3DTransform(start3D);

    start.x = lcd.x;
    start.y = lcd.y;

    lcd = get3DTransform(end3D);

    end.x = lcd.x;
    end.y = lcd.y;

    /*c3D.x_value = start3D.x_value;
    c3D.y_value = end3D.y_value + (lambda*(end3D.y_value - start3D.y_value));
    c3D.z_value = end3D.z_value + (lambda*(end3D.z_value - start3D.z_value));*/

    c3D.x_value = end3D.x_value + (lambda*(end3D.x_value - start3D.x_value));
    c3D.y_value = start3D.y_value;
    c3D.z_value = end3D.z_value + (lambda*(end3D.z_value - start3D.z_value));

    lcd = get3DTransform(c3D);

    c.x = lcd.x;
    c.y = lcd.y;

    drawLine(c.x, c.y, end.x, end.y, GREEN);
    designTreeIn3D(end3D, c3D, level - 1, lambda);

    rtl3D = rotate_pointIn3D(c3D, end3D, pi/6);

    lcd = get3DTransform(rtl3D);

    drawLine(lcd.x, lcd.y, end.x, end.y, GREEN);
    designTreeIn3D(end3D, rtl3D, level - 1, lambda);

    rtr3D = rotate_pointIn3D(c3D, end3D, -pi/6);

    lcd = get3DTransform(rtr3D);

    drawLine(lcd.x, lcd.y, end.x, end.y, GREEN);
    designTreeIn3D(end3D, rtr3D, level - 1, lambda);
}

//Rotate cube with respect to the arbitrary vector
Pts3D rotateCoord3D(Pts3D ARBPi, Pts3D ARBPi1, int angle, float cube_x, float cube_y, float
cube_z)
{

```

```

Pts3D ARBPi1_T, ARBPi1_R;
Pts3D cube_T, cube_Rzw, cube_Ryw, cube_Rzw_main, cube_Ryw_reverse, cube_Rzw_reverse,
cube_Treverse;

float cos_alpha_dash, sin_alpha_dash, cos_alpha_doubledash, sin_alpha_doubledash;

float denom1, denom2;

float angle_rad;

//Step1: Translation
//Find new Pi+1 point after Translation for calculating cos(alpha_dash) and
sin(alpha_dash) values
ARBPi1_T.x_value = ARBPi1.x_value - ARBPi.x_value;
ARBPi1_T.y_value = ARBPi1.y_value - ARBPi.y_value;
ARBPi1_T.z_value = ARBPi1.z_value - ARBPi.z_value;

//Multiply Translation matrix with the given cube point
cube_T.x_value = cube_x - ARBPi.x_value;
cube_T.y_value = cube_y - ARBPi.y_value;
cube_T.z_value = cube_z - ARBPi.z_value;

//Step2: Rotation with respect to Zw axis (Note that here alpha_dash will be negative
since rotation is clockwise => so this changes the rotation_Zw matrix)
//Calculate cos(alpha_dash) and sin(alpha_dash) values
denom1 = sqrt(pow(ARBPi1_T.x_value,2) + pow(ARBPi1_T.y_value,2));
cos_alpha_dash = ARBPi1_T.x_value / denom1;
sin_alpha_dash = ARBPi1_T.y_value / denom1;

//Find new Pi+1 point after Rotation. This new point is used to calculate the
cos(alpha_doubledash) and sin(alpha_doubledash) values
ARBPi1_R.x_value = ARBPi1_T.x_value*cos_alpha_dash + ARBPi1_T.y_value*sin_alpha_dash;
ARBPi1_R.y_value = ARBPi1_T.y_value*cos_alpha_dash - ARBPi1_T.x_value*sin_alpha_dash;
ARBPi1_R.z_value = ARBPi1_T.z_value;

//Multiply Rotation_Zw matrix with the translated cube point
cube_Rzw.x_value = cube_T.x_value*cos_alpha_dash + cube_T.y_value*sin_alpha_dash;
cube_Rzw.y_value = cube_T.y_value*cos_alpha_dash - cube_T.x_value*sin_alpha_dash;
cube_Rzw.z_value = cube_T.z_value;

//Step3: Rotation with respect to Yw axis (Note that here alpha_doubledash will be
negative since rotation is clockwise => so this changes the rotation_Yw matrix)
//Calculate cos(alpha_dash) and sin(alpha_dash) values
denom2 = sqrt(pow(ARBPi1_R.x_value,2) + pow(ARBPi1_R.z_value,2));
cos_alpha_doubledash = ARBPi1_R.z_value / denom2;
sin_alpha_doubledash = ARBPi1_R.x_value / denom2;

//Multiply Rotation_Yw matrix with the rotated_Zw cube point
cube_Ryw.x_value = cube_Rzw.x_value*cos_alpha_doubledash -
cube_Rzw.z_value*sin_alpha_doubledash;
cube_Ryw.y_value = cube_Rzw.y_value;
cube_Ryw.z_value = cube_Rzw.z_value*cos_alpha_doubledash +
cube_Rzw.x_value*sin_alpha_doubledash;

//Step4: Rotation with respect to Zw axis
//change the given angle from degrees to radians
angle_rad = (pi * angle) / 180;

//Multiply Rotation_Yw matrix with the rotated_Yw cube point
cube_Rzw_main.x_value = cube_Ryw.x_value*cos(angle_rad) -
cube_Ryw.y_value*sin(angle_rad);
cube_Rzw_main.y_value = cube_Ryw.y_value*cos(angle_rad) +
cube_Ryw.x_value*sin(angle_rad);
cube_Rzw_main.z_value = cube_Ryw.z_value;

//Step5: Revert back the Rotation with respect to Yw axis
//Multiply Rotation_Yw matrix with the rotated_Zw_main cube point

```

```

    cube_Ryw_reverse.x_value = cube_Rzw_main.x_value*cos_alpha_doubledash +
cube_Rzw_main.z_value*sin_alpha_doubledash;
    cube_Ryw_reverse.y_value = cube_Rzw_main.y_value;
    cube_Ryw_reverse.z_value = cube_Rzw_main.z_value*cos_alpha_doubledash -
cube_Rzw_main.x_value*sin_alpha_doubledash;

    //Step6: Revert back the Rotation with respect to Z_w axis
    //Multiply Rotation_Yw matrix with the rotated_Yw_reverse cube point
    cube_Rzw_reverse.x_value = cube_Ryw_reverse.x_value*cos_alpha_dash -
cube_Ryw_reverse.y_value*sin_alpha_dash;
    cube_Rzw_reverse.y_value = cube_Ryw_reverse.y_value*cos_alpha_dash +
cube_Ryw_reverse.x_value*sin_alpha_dash;
    cube_Rzw_reverse.z_value = cube_Ryw_reverse.z_value;

    //Step7: Revert back the Translation
    //Multiply Translation matrix with the given cube point
    cube_Treverse.x_value = cube_Rzw_reverse.x_value + ARBPi.x_value;
    cube_Treverse.y_value = cube_Rzw_reverse.y_value + ARBPi.y_value;
    cube_Treverse.z_value = cube_Rzw_reverse.z_value + ARBPi.z_value;

    return cube_Treverse;
}

//Draw the cube
void drawCube()
{
    #define UpperBD 52
    #define NumOfPts 16
    typedef struct{
        float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
    }pworld;
    typedef struct{
        float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
    }pviewer;
    typedef struct{
        float X[UpperBD]; float Y[UpperBD];
    }pperspective;

    // EYE VIEW(200,200,200),

    float Xe=150, Ye=150, Ze=100;

    float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Ze,2));
    float D_focal=120; //For perspective projection

    pworld WCS;
    pviewer V;
    pperspective P;
    Pts3D Vsingle, RWCS, ARBPi, ARBPi1;
    Pts2D Psingle;
    float_t L1,L2,L3,L4;
    int angle, i;

    angle = -5; //minus for clockwise

    // X-Y-Z World Coordinate System with each of the axis equal to 150

    // Origin
    WCS.X[0]=0.0; WCS.Y[0]=0.0; WCS.Z[0]=0.0;

    //Axes
    WCS.X[1]=150.0; WCS.Y[1]=0.0; WCS.Z[1]=0.0;
    WCS.X[2]=0.0; WCS.Y[2]=150.0; WCS.Z[2]=0.0;
    WCS.X[3]=0.0; WCS.Y[3]=0.0; WCS.Z[3]=150.0;

    // Elevate Cube along Z_w axis by 10
    // New points to define the cube center as (80,80,35)

```

```

WCS.X[4]=55.0; WCS.Y[4]=55.0; WCS.Z[4]=10.0;
WCS.X[5]=55.0; WCS.Y[5]=55.0; WCS.Z[5]=60.0;
WCS.X[6]=55.0; WCS.Y[6]=105.0; WCS.Z[6]=10.0;
WCS.X[7]=55.0; WCS.Y[7]=105.0; WCS.Z[7]=60.0;
WCS.X[8]=105.0; WCS.Y[8]=55.0; WCS.Z[8]=10.0;
WCS.X[9]=105.0; WCS.Y[9]=55.0; WCS.Z[9]=60.0;
WCS.X[10]=105.0; WCS.Y[10]=105.0; WCS.Z[10]=10.0;
WCS.X[11]=105.0; WCS.Y[11]=105.0; WCS.Z[11]=60.0;

//Define given Arbitrary vectors
ARBPi.x_value=0.0; ARBPi.y_value=0.0; ARBPi.z_value=35.0;
ARBPi1.x_value=200.0; ARBPi1.y_value=220.0; ARBPi1.z_value=40.0;

for(i=4;i<12;i++)
{
    RWCS = rotateCoord3D(ARBPi, ARBPi1, angle, WCS.X[i], WCS.Y[i], WCS.Z[i]);
    WCS.X[i] = RWCS.x_value;
    WCS.Y[i] = RWCS.y_value;
    WCS.Z[i] = RWCS.z_value;
}

//Shadow Calculation

// Point of Light Source PS(-20, -20, 220)
WCS.X[12]=-20.0; WCS.Y[12]=-20.0; WCS.Z[12]=220.0;

Pts3D Ps;
Ps.x_value = WCS.X[12]; Ps.y_value = WCS.Y[12]; Ps.z_value = WCS.Z[12];

Pts3D P1;
P1.x_value = WCS.X[5]; P1.y_value = WCS.Y[5]; P1.z_value = WCS.Z[5];

Pts3D P2;
P2.x_value = WCS.X[9]; P2.y_value = WCS.Y[9]; P2.z_value = WCS.Z[9];

Pts3D P3;
P3.x_value = WCS.X[11]; P3.y_value = WCS.Y[11]; P3.z_value = WCS.Z[11];

Pts3D P4;
P4.x_value = WCS.X[7]; P4.y_value = WCS.Y[7]; P4.z_value = WCS.Z[7];

// Shadow Points

Pts3D S1,S2,S3,S4;
L1 = Lambda3D(WCS.Z[5],WCS.Z[12]);
L2 = Lambda3D(WCS.Z[9],WCS.Z[12]);
L3 = Lambda3D(WCS.Z[11],WCS.Z[12]);
L4 = Lambda3D(WCS.Z[7],WCS.Z[12]);
S1 = ShadowPoint3D(P1,Ps,L1);
S2 = ShadowPoint3D(P2,Ps,L2);
S3 = ShadowPoint3D(P3,Ps,L3);
S4 = ShadowPoint3D(P4,Ps,L4);

WCS.X[13]=S1.x_value;      WCS.Y[13]=S1.y_value;      WCS.Z[13]=S1.z_value;
//S1
WCS.X[14]=S2.x_value;      WCS.Y[14]=S2.y_value;      WCS.Z[14]=S2.z_value;
//S2
WCS.X[15]=S3.x_value;      WCS.Y[15]=S3.y_value;      WCS.Z[15]=S3.z_value;
//S3
WCS.X[16]=S4.x_value;      WCS.Y[16]=S4.y_value;      WCS.Z[16]=S4.z_value;
//S4

float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
float cPhi = Ze/Rho;

```



```

// WORLD TO VIEWER TRANSFORM

for(int i=0;i<=NumOfPts;i++)
{
    V.X[i] = -sPheta * WCS.X[i] + cPheta * WCS.Y[i];
    V.Y[i] = -cPheta * cPhi * WCS.X[i] - cPhi * sPheta * WCS.Y[i] + sPhi *
WCS.Z[i];
    V.Z[i] = -sPhi * cPheta * WCS.X[i] - sPhi * cPheta * WCS.Y[i] - cPhi * WCS.Z[i]
+ Rho;
}

for(int i=0;i<=NumOfPts;i++)
{
    P.X[i]=V.X[i]*(D_focal/V.Z[i]);
    P.Y[i]=V.Y[i]*(D_focal/V.Z[i]);
}

Pts3D temp, temp1;

temp.x_value = WCS.X[5]; temp.y_value = WCS.Y[5]; temp.z_value = WCS.Z[5];

float temp_color, temp_color1;

temp_color = getDiffuseColor(temp);

// Draw Lines for all the edges of the cube
drawLine(P.X[0],P.Y[0],P.X[1],P.Y[1],RED);
drawLine(P.X[0],P.Y[0],P.X[2],P.Y[2],0x00FF00);
drawLine(P.X[0],P.Y[0],P.X[3],P.Y[3],0x0000FF);

//New Centered Cube DrawLines
drawLine(P.X[6],P.Y[6],P.X[4],P.Y[4],WHITE);
drawLine(P.X[10],P.Y[10],P.X[8],P.Y[8],WHITE);
drawLine(P.X[6],P.Y[6],P.X[10],P.Y[10],BLUE);
drawLine(P.X[8],P.Y[8],P.X[4],P.Y[4],WHITE);

drawLine(P.X[7],P.Y[7],P.X[5],P.Y[5],temp_color);
drawLine(P.X[7],P.Y[7],P.X[11],P.Y[11],WHITE);
drawLine(P.X[9],P.Y[9],P.X[11],P.Y[11],WHITE);
drawLine(P.X[9],P.Y[9],P.X[5],P.Y[5],temp_color);

drawLine(P.X[9],P.Y[9],P.X[8],P.Y[8],temp_color);
drawLine(P.X[11],P.Y[11],P.X[10],P.Y[10],WHITE);
drawLine(P.X[5],P.Y[5],P.X[4],P.Y[4],temp_color);
drawLine(P.X[7],P.Y[7],P.X[6],P.Y[6],BLUE);

// Shadow Drawlines
drawLine(P.X[13],P.Y[13],P.X[14],P.Y[14],DARKBLUE);
drawLine(P.X[14],P.Y[14],P.X[15],P.Y[15],DARKBLUE);
drawLine(P.X[16],P.Y[16],P.X[15],P.Y[15],DARKBLUE);
drawLine(P.X[16],P.Y[16],P.X[13],P.Y[13],DARKBLUE);

double xvaluestart = S1.x_value;
double xvalueend = S2.x_value;
double yvaluestart = S1.y_value;
double yvalueend = S4.y_value;

/*printf("%lf\n", xvaluestart);
printf("%lf\n", xvalueend);
printf("%lf\n", yvaluestart);
printf("%lf\n", yvalueend);*/

//Shadow fill
for(double x=xvaluestart;x<=xvalueend;x+=1.234)
    for(double y=yvaluestart;y<=yvalueend;y+=1.234)
    {
        Pts3D pt; Pts2D pt_new;

```

```

        pt.x_value=x; pt.y_value=y; pt.z_value=0;
        pt_new = get3DTransform(pt);
        drawPixel(pt_new.x,pt_new.y,DARKBLUE);
    }

//Red - top side Fill
float diff_color;
for(float y=WCS.Y[5];y<=WCS.Y[6];y+=0.9888)
    for(float x=WCS.X[4];x<=WCS.X[8];x+=0.9888)
    {
        Pts3D pt; Pts2D pt2d;
        pt.x_value=x; pt.y_value=y; pt.z_value=WCS.Z[5];
        pt2d = get3DTransform(pt);
        // RED DIFFUSE
        diff_color = getDiffuseColor(pt);
        drawPixel(pt2d.x,pt2d.y,diff_color);
    }

//Green - left side Fill
for(float y=WCS.Y[5];y<=WCS.Y[6];y+=0.988)
    for(float z=WCS.Z[5];z>=WCS.Z[4];z-=0.988)
    {
        Pts3D pt; Pts2D pt2d;
        pt.x_value=WCS.X[8]; pt.y_value=y; pt.z_value=z;
        pt2d = get3DTransform(pt);
        drawPixel(pt2d.x,pt2d.y,GREEN);
    }

//Blue - right side Fill
for(float z=WCS.Z[5];z>=WCS.Z[4];z-=0.988)
    for(float x=WCS.X[8];x>=WCS.X[4];x-=0.988)
    {
        Pts3D pt; Pts2D pt2d;
        pt.x_value=x; pt.y_value=WCS.Y[6]; pt.z_value=z;
        pt2d = get3DTransform(pt);
        drawPixel(pt2d.x,pt2d.y,BLUE);
    }

int cube_side = 50;

drawTree(WCS.X[4], WCS.Y[4], WCS.Z[4], cube_side);
}

// Method to draw the half sphere using contours
void drawSphere()
{
    #define UpperBD 360
    #define UpperPCBD 800
    #define TotalPts 360
    #define NumOfLevels 20
    typedef struct{
        float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
    }pworld;
    typedef struct{
        float X[UpperBD]; float Y[UpperBD]; float Z[UpperBD];
    }pviewer;
    typedef struct{
        float X[UpperBD]; float Y[UpperBD];
    }pperspective;

    // Structure to hold the equi-distant points on each contour
    typedef struct{
        float X[UpperPCBD]; float Y[UpperPCBD]; float Z[UpperPCBD];
    }pcontour;

    pworld WCS;
    pviewer V;
    pperspective P;
    pcontour PC;
    Pts3D Vsingletemp, temp3D;

```

```

Pts2D Psingle;

float Xe=150, Ye=150, Ze=100;

float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Ze,2));
float D_focal=120;

int diffColor[UpperBD];
int diffColorPC[UpperPCBD];

int i,level;
float y;

float angle_theta;
int radius = 100;
int delta = 10;
int kx=0, ky=0, kz=0;

for(level=0;level<=NumOfLevels-1;level++) //20 levels of cross section contours
{
    for(i=0;i<TotalPts;i++)
    {
        // Logic to draw a circle using angles
        angle_theta = i*3.142 /180;
        WCS.X[i] = 0 + radius*cos(angle_theta);
        WCS.Y[i] = 0 + radius*sin(angle_theta);
        WCS.Z[i] = 4*level; //Contours one above the other with the distance of
10.

        temp3D.x_value = WCS.X[i];
        temp3D.y_value = WCS.Y[i];
        temp3D.z_value = WCS.Z[i];
        diffColor[i] = getDiffuseColorGreen(temp3D);
    }

    float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
    float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
    float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
    float cPhi = Ze/Rho;

    // WORLD TO VIEWER TRANSFORM

    for(i=0;i<TotalPts;i++)
    {
        V.X[i] = -sPheta * WCS.X[i] + cPheta * WCS.Y[i];
        V.Y[i] = -cPheta * cPhi * WCS.X[i] - cPhi * sPheta * WCS.Y[i] + sPhi *
WCS.Z[i];
        V.Z[i] = -sPhi * cPheta * WCS.X[i] - sPhi * cPheta * WCS.Y[i] - cPhi *
WCS.Z[i] + Rho;
    }

    // Viewer to perspective transform for each point
    for(i=0;i<TotalPts;i++)
    {
        P.X[i]=V.X[i]*(D_focal/V.Z[i]);
        P.Y[i]=V.Y[i]*(D_focal/V.Z[i]);
        drawPixel(P.X[i], P.Y[i], diffColor[i]);

        // Logic to store a selected number of equi-distant points on each
contour into PC.
        // In this case, for each contour, 24 points are selected and later
joined.
        if(i%(TotalPts/40) == 0)
        {
            PC.X[kx++] = P.X[i];
            PC.Y[ky++] = P.Y[i];

```

```

        diffColorPC[kz++] = diffColor[i];
    }
}

//radius-=10;
//radius -= (level + 1);
radius-=0.25*level;    // decrease radius of each contour when level increases
and Z_w increases
}

// Logic to join the equi-distant points of one contour to the next contour
for(i=0;i<kx;i++)
{
    if((i+40)<kx)
        drawLine(PC.X[i], PC.Y[i], PC.X[i+40], PC.Y[i+40], diffColorPC[i]);
}
}

int main (void)
{
    uint32_t pnum = 0 ;

    if ( pnum == 0 )
        SSP0Init();
    else
        puts("Port number is incorrect");

    lcd_init();

    fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);

    drawSphere();

    drawCube();

    return 0;
}

```